Acceleration of the shiftable $O(1)$ algorithm for bilateral filtering and non-local means

Kunal N. Chaudhury*

March 1, 2013

Abstract

A direct implementation of the bilateral filter [1] requires $O(\sigma_s^2)$ operations per pixel, where σ_s is the (effective) width of the spatial kernel. A fast implementation of the bilateral filter was recently proposed in [19] that required $O(1)$ operations per pixel with respect to σ_s. This was done by using trigonometric functions for the range kernel of the bilateral filter, and by exploiting their so-called *shiftability* property. In particular, a fast implementation of the Gaussian bilateral filter was realized by approximating the Gaussian range kernel using raised cosines. Later, it was demonstrated in [24] that this idea could be extended to a larger class of filters, including the popular non-local means filter [2, 3]. As already observed in [19], a flip side of this approach was that the run time depended on the width σ_r of the range kernel. For an image with dynamic range $[0, T]$, the run time scaled as $O(T^2/\sigma_r^2)$ with σ_r. This made it difficult to implement narrow range kernels, particularly for images with large dynamic range. In this paper, we discuss this problem, and propose some simple steps to accelerate the implementation, in general, and for small σ_r in particular. We provide some experimental results to demonstrate the acceleration that is achieved using these modifications.

Keywords: Bilateral filter, non-local means, shiftability, constant-time algorithm, Gaussian kernel, truncation, running maximum, max filter, recursive filter, $O(1)$ complexity.

*Correspondence: kchaudhu@math.princeton.edu.

1 Introduction

The bilateral filter is an edge-preserving diffusion filter, which was introduced by Tomasi et al. in [1]. The edge-preserving property comes from the use of a range kernel (along with the spatial kernel) that is used to control the diffusion in the vicinity of edges. In this work, we will focus on the Gaussian bilateral filter where both the spatial and range kernels are Gaussian [1]. This is given by

$$\tilde{f}(\boldsymbol{x}) = \frac{1}{\eta} \int_{\Omega} g_{\sigma_s}(\boldsymbol{x} - \boldsymbol{y})\, g_{\sigma_r}(f(\boldsymbol{x} - \boldsymbol{y}) - f(\boldsymbol{x}))\, f(\boldsymbol{x} - \boldsymbol{y})\, d\boldsymbol{y} \tag{1}$$

where

$$\eta = \int_{\Omega} g_{\sigma_s}(\boldsymbol{x} - \boldsymbol{y})\, g_{\sigma_r}(f(\boldsymbol{x} - \boldsymbol{y}) - f(\boldsymbol{x}))\, d\boldsymbol{y}.$$

Here, $g_{\sigma_s}(\boldsymbol{x})$ is the centered Gaussian distribution on the plane with variance σ_s^2, and $g_{\sigma_r}(s)$ is the one-dimensional Gaussian distribution with variance σ_r^2; Ω is the support of $g_{\sigma_s}(\boldsymbol{x})$ over which the averaging takes place. We call $g_{\sigma_s}(\boldsymbol{x})$ and $g_{\sigma_r}(s)$ the spatial and the range kernel.

The range kernel is controlled by the local distribution of intensity. Sharp discontinuities (jumps) in intensity typically occur in the vicinity of edges. This is picked up by the range kernel, which is then used to inhibit the spatial diffusion. On the other hand, the range kernel becomes inoperative in regions with smooth variations in intensity. The spatial kernel then takes over, and the bilateral filter behaves as a standard diffusion filter. Together, the spatial and range kernels perform smoothing in homogeneous regions, while preserving edges at the same time [1].

The bilateral filter has found widespread use in several image processing, computer graphics, and computer vision applications [4, 5, 6, 8, 9]; see [10] for further applications. More recently, the bilateral filter was extended by Baudes et al. [2] in the form of the non-local means filter, where the similarity between pixels is measured using patches centered around the pixel.

1.1 Fast bilater filter

The direct implementation of (1) is computationally intensive, especially when σ_s is large (σ_r has no effect on the run time in this case). In particular, the direct implementation requires $O(\sigma_s^2)$ operations per pixel. This makes the filter slow for real-time applications. Several efficient algorithms have been proposed in the past for implementing the filter in real time, e.g., see

[11, 12, 15, 13, 26, 27]. In [14], Porikli demonstrated for the first time that the bilateral filter could be implemented using $O(1)$ operations per pixel (with respect to σ_s). This was done for two different settings: (a) Spatial box filter and arbitrary range filter, and (b) Arbitrary spatial filter and polynomial range filter. The author extended (b) to the Gaussian bilateral filter in (1) by approximating $g_{\sigma_r}(s)$ with its Taylor polynomial. The run time of this approximation was linear in the order of the polynomial. The problem with Taylor polynomials, however, is that they provide good approximations of $g_{\sigma_r}(s)$ only locally around the origin. In particular, they have the following drawbacks:

- Taylor polynomials are not guaranteed to be positive and monotonic away from the origin, where the approximation is poor. Moreover, they tend to blow up at the tails.
- It is difficult to approximate $g_{\sigma_r}(s)$ using the Taylor expansion when σ_r is small. In particular, a large order polynomial is required to get a good approximation of a narrow Gaussian, and this considerably increases the run time of the algorithm.

The first of these problems was addressed in [19]. In this paper, the authors observed that it is important that the kernel used to approximate $g_{\sigma_r}(s)$ be positive, monotonic, and symmetric. While it is easy to ensure symmetry, the other two properties are hard to enforce using Taylor approximations. It was noticed that, in the absence of these properties, the bilateral filter in [14] created strange artifacts in the processed image (cf. Figure 3 in [19]). The authors proposed to fix this problem using the family of raised cosines, namely, functions of the form

$$\phi(s) = \Big[\cos\Big(\frac{\pi s}{2T}\Big)\Big]^N \qquad (-T \le s \le T). \tag{2}$$

Here N is the order of the kernel, which controls the width of $\phi(s)$. The kernel can be made narrow by increasing N.

The key parameter in (2) is the quantity T. The idea here is that $[\cos(s)]^N$ is guaranteed to be positive and monotonic provided that s is restricted to the interval $[-\pi/2, \pi/2]$. Note that the argument s in (2) takes on the values $|f(\boldsymbol{x}-\boldsymbol{y}) - f(\boldsymbol{x})|$ as $\boldsymbol{x}$ and $\boldsymbol{y}$ varies over the image. Therefore, by letting

$$T = \max_{\boldsymbol{x}} \max_{\boldsymbol{y}\in\Omega} |f(\boldsymbol{x}-\boldsymbol{y}) - f(\boldsymbol{x})|,$$

one could guarantee $\phi(s)$ to be positive and monotonic over $[-T, T]$. In [19], T was simply set to the maximum dynamic range, for example, 255 for

grayscale images. We refer the readers to Figure 2 in [19] for a comparison of (2) and the polynomial kernels used in [14]. It was later observed in [24] that polynomials could also be used for the same purpose. The polynomials suggested were of the form

$$\phi(s) = \left(1 - \frac{s^2}{T^2}\right)^N \qquad (-T \leq s \leq T). \tag{3}$$

1.2 Fast $O(1)$ implementation using shiftable kernels

For completeness, we now explain how the above kernels can be used to compute (1) using $O(1)$ operations. As observed in [24], (2) and (3) are essentially the simplest kernels that have the so-called property of *shiftability*. This means that, for a given N, we can find a fixed set of basis functions $\phi_1(s), \ldots, \phi_N(s)$ and coefficients $c_1, \ldots, c_N$, so that for any translation τ, we can write

$$\phi(s - \tau) = c_1(\boldsymbol{\tau})\phi_1(s) + \cdots + c_N(\boldsymbol{\tau})\phi_N(s). \tag{4}$$

The coefficients depend continuously on τ, but the basis functions have no dependence on τ. For (2), both the basis functions and coefficients are cosines, while they are polynomials for (3). This shiftability property is at the heart of the $O(1)$ algorithm. Let $\overline{f}(\boldsymbol{x})$ denote the output of the Gaussian filter $g_{\sigma_s}(\boldsymbol{x})$ with neighborhood Ω,

$$\overline{f}(\boldsymbol{x}) = \int_\Omega g_{\sigma_s}(\boldsymbol{x} - \boldsymbol{y}) f(\boldsymbol{y})\, d\boldsymbol{y}. \tag{5}$$

Note that, by replacing $g_{\sigma_r}(s)$ with $\phi(s)$, we can write (1) as

$$\tilde{f}(\boldsymbol{x}) = \frac{1}{\eta}\Big[c_1(f(\boldsymbol{x}))\, \overline{F_1}(\boldsymbol{x}) + \cdots c_N(f(\boldsymbol{x}))\, \overline{F_N}(\boldsymbol{x})\Big], \tag{6}$$

where we have set $F_i(\boldsymbol{x}) = f(\boldsymbol{x})\phi_i(f(\boldsymbol{x}))$. Similarly, by setting $G_i(\boldsymbol{x}) = \phi_i(f(\boldsymbol{x}))$, we can write

$$\eta = c_1(f(\boldsymbol{x}))\, \overline{G_1}(\boldsymbol{x}) + \cdots + c_N(f(\boldsymbol{x}))\, \overline{G_N}(\boldsymbol{x}). \tag{7}$$

Now, it is well-known that certain approximation of (5) can be computed using just $O(1)$ operations per pixel. These recursive algorithms are based on specialized kernels, such as the box and the hat function [21, 23], and the more general class of box splines [16]. Putting all these together, we arrive at the following $O(1)$ algorithm for approximating (1):

1. Fix N, and approximate $g_{\sigma_r}(s)$ using (2) or (3).
2. For $i = 1, 2, \ldots, N$, set up the images $F_i(\boldsymbol{x}) = f(\boldsymbol{x})\phi_i(f(\boldsymbol{x}))$ and $G_i(\boldsymbol{x}) = \phi_i(f(\boldsymbol{x}))$, and the coefficients $c_i(f(\boldsymbol{x}))$.
3. Use a recursive $O(1)$ algorithm to compute each $\overline{F_i}(\boldsymbol{x})$ and $\overline{G_i}(\boldsymbol{x})$.
4. Plug these into (6) and (7) to get the filtered image.

It is clear that better approximations are obtained when N is large. On the other hand, the run time scales linearly with N. One key advantage of the above algorithm, however, is that the $\overline{F_i}(\boldsymbol{x})$ and $\overline{G_i}(\boldsymbol{x})$ can be computed in parallel. For small orders ($N < 10$), the serial implementation is found to be comparable, and often better, than the state-of-the-art algorithms. The parallel implementation, however, turns out to be much faster than the competing algorithms, at least for $N < 50$. Henceforth, we will refer to the above algorithm as `SHIFTABLE-BF`, the shiftable bilateral filter.

1.3 Gaussian approximation for small σ_r

This brings us to the question as to whether we can always work with, say, $N < 50$ basis functions, in `SHIFTABLE-BF`? To answer this question, we must explain in some detail step (1) of the algorithm, where we approximate the Gaussian range kernel,

$$g_{\sigma_r}(s) = \exp\left(-\frac{s^2}{2\sigma_r^2}\right),$$

on the interval $[-T, T]$. This could be done either using (2),

$$g_{\sigma_r}(s) = \lim_{N \longrightarrow \infty} \left[\cos\left(\frac{s}{\sqrt{N}\sigma}\right)\right]^N, \tag{8}$$

or, using (3),

$$g_{\sigma_r}(s) = \lim_{N \longrightarrow \infty} \left(1 - \frac{s^2}{2N\sigma^2}\right)^N. \tag{9}$$

These approximations were proposed in [19, 24]. Note that, we have to rescale (2) by $\sqrt{N}$, and (3) by N, to get to the right limit. On the other hand, to ensure positivity and monotonicity, we need to guarantee that the arguments of (8) and (9) are in the intervals $[-\pi/2, \pi/2]$ and $[0, 1]$. A simple calculation shows that this is the case provided that N is larger than

$N_0 = 4T^2/\pi^2\sigma_r^2 = 0.405(T/\sigma_r)^2$ for the former, and $N_0 = 0.5(T/\sigma_r)^2$ for the latter. In other words, it is not sufficient to set N large – it must be at least be as large as N_0. In Table 1, we give the values of N_0 for different values of σ_r when $T = 255$. It is seen that N_0 gets impracticable large for $\sigma_r < 30$. This does not come as a surprise since it is well-known that one requires a large number of trigonometric functions (or polynomials) to closely approximate a narrow Gaussian on a large interval. As pointed out earlier, this was also one of the problems in [14].

Table 1: The threshold N_0 for different σ_r ($T = 255$).

σ_r	5	10	20	30	40	60	80	100
N_0	1053	263	66	29	16	7	4	3

1.4 Present Contributions

In this paper, we address the above problem, namely that N_0 grows as $O(T^2/\sigma_r^2)$ with σ_r. In Section 2, we propose a fast algorithm for determining T exactly. Besides cutting down N_0, this is essential for determining the (local) dynamic range of a grayscale image that has been deformed, e.g., by additive noise. Setting $T = 255$ in this case can lead to artifacts in the processed image. Next, in Section 3, we provide a simple and practical means of reducing the order, which leads to quite dramatic reductions in the run time of `SHIFTABLE-BF`. These modifications are also applicable to the shiftable algorithms proposed in [19, 24]. Finally, in Section 4, we provide some experimental results to demonstrate the acceleration that is achieved using these modifications. We also compare our algorithm with the Porikli's algorithms [14], both in terms of speed and accuracy.

2 Fast algorithm for finding T

For the rest of the discussion, we work with finite-sized images (bounded Ω) on the Cartesian grid. We continue to use $\boldsymbol{x}$ and $\boldsymbol{y}$ to denote points on the grid. The integral in (1) is simply replaced by a finite sum over Ω. We will use the norm $\|\boldsymbol{x}\| = |x_1|+|x_2|$, where $\boldsymbol{x} = (x_1, x_2)$. Without loss of generality, we assume that Ω is a square neighborhood, that is, $\Omega = \{\boldsymbol{x} : \|\boldsymbol{x}\| \leq R\}$ where, say, $R = 3\sigma_s$ (if Ω is not rectangular, we take the smallest rectangle containing Ω).

Note that, for a given σ_r, we can cut down N_0 by using a tight estimate for

$$T = \max_{\boldsymbol{x}} \max_{\|\boldsymbol{y}\| \leq R} |f(\boldsymbol{x} - \boldsymbol{y}) - f(\boldsymbol{x})|. \tag{10}$$

The smaller the estimate, the lower is the threshold N_0. The point is that the worst-case estimate $T = 255$ is often rather loose for grayscale images. For example, we give the exact values of T for a test image in Table 2, computed at different values of σ_s. We also give the time required to compute T.

Table 2: Exact values of T for the standard 512×512 *Lena*. Also shown is the time needed to compute T using Matlab.

σ_s	1	3	5	10	15	20	30
T	153	205	208	210	211	215	215
time (sec)	2.06	2.13	2.33	2.71	3.26	4.01	5.60

It is seen that the exact values of T are indeed much less than the worst-case estimate, particularly for small σ_s. For $\sigma_s = 3$, T is only about 205. Even for σ_s as large as 30, T is about 215. Consider the bilateral filter with $\sigma_s = 10$ and $\sigma_r = 10$. From Table 1, $N_0 = 263$ using $T = 255$. However, using the exact value $T = 210$, we can bring this down to $263 \cdot (210/255)^2 \approx 178$, a reduction by almost 100. For smaller values of σ_r, this gain is even more drastic. However, notice the time required to compute T in Table 1. This increases quickly with the increase in σ_s (in fact, scales as $O(R^2)$). Experiments show us that, for large σ_s, this is comparable to the time required to compute the bilateral filter. It would thus help to have an $O(1)$ algorithm for computing T. Motivated by our previous work on filtering using running sums [16], we recently devised an algorithm that does exactly this. We later found that the algorithm had already been discovered two decades back in a different context [17, 18].

Our algorithm is based on the following observations. First, note that we can take out the modulus from (10) using symmetry.

Proposition 2.1 (Simplification)**.**

$$T = \max_{\boldsymbol{x}} \left[f(\boldsymbol{x}) - \max_{\|\boldsymbol{y}\| \leq R} f(\boldsymbol{x} - \boldsymbol{y}) \right]. \tag{11}$$

Proof. This follows from the observations that $|t| = \max(t, -t)$, and that $\|\boldsymbol{x} - \boldsymbol{y}\| \leq R$ is symmetric in $\boldsymbol{x}$ and $\boldsymbol{y}$. Moreover, note that the operation that takes two numbers a and b and returns $\max(a, b)$ is associative. Using

associativity, we can write (10) as $T = \max(T_+, T_-)$, where

$$T_+ = \max_{\boldsymbol{x}} \left[f(\boldsymbol{x}) - \max_{\|\boldsymbol{y}\| \leq R} f(\boldsymbol{x} - \boldsymbol{y}) \right], \tag{12}$$

and

$$T_- = \max_{\boldsymbol{x}} \left[\max_{\|\boldsymbol{y}\| \leq R} f(\boldsymbol{x} - \boldsymbol{y}) - f(\boldsymbol{x}) \right], \tag{13}$$

We claim that $T_+ = T_-$, so that we need not compute them separately. Indeed, suppose that the first maximum is attained at $\boldsymbol{x}_0$ and $\boldsymbol{y}_0$, that is, $T_+ = f(\boldsymbol{x}_0) - f(\boldsymbol{x}_0 - \boldsymbol{y}_0)$. Taking $\boldsymbol{x} = \boldsymbol{x}_0 - \boldsymbol{y}_0$ and $\boldsymbol{y} = -\boldsymbol{y}_0$, and noting that $\|\boldsymbol{x} - \boldsymbol{y}\| \leq R$, we must have

$$T_- \geq f(\boldsymbol{x} - \boldsymbol{y}) - f(\boldsymbol{x}) = f(\boldsymbol{x}_0) - f(\boldsymbol{x}_0 - \boldsymbol{y}_0) = T_+.$$

By an identical argument, $T_+ \geq T_-$, and the proposition follows. □

The problem is now reduced to that of computing the windowed maximums in (11). A direct computation would still require $O(R^2)$ comparisons. It turns out that we can do this very fast (no matter how large is R) by exploiting the overlap between adjacent windows. This is done by adapting the so-called `MAX-FILTER` algorithm.

Proposition 2.2 (Max-Filter Algorithm)**.** *There is an $O(1)$ algorithm for computing*

$$\max_{\|\boldsymbol{y}\| \leq R} f(\boldsymbol{x} - \boldsymbol{y})$$

at every $\boldsymbol{x}$.

This algorithm was first proposed by van Herk, and Gil and Werman [17, 18]. It is clear that since the search domain Ω is separable, it suffices to solve the problem in one dimension. The problem in two-dimensions can be solved simply by iterating the one-dimensional `MAX-FILTER` along each dimension. From (11), we arrive at Algorithm 1 for computing T with $O(1)$ operations. We note that Algorithm 1 does not actually compute $\max\{|f(\boldsymbol{x} - \boldsymbol{y}) - f(\boldsymbol{x})| : \|\boldsymbol{y}\| \leq R\}$ at every $\boldsymbol{x}$. It only has access to the distribution of the maximums. For completeness, we have explained the `MAX-FILTER` algorithm in the Appendix. For further details, we refer the readers to [17, 18].

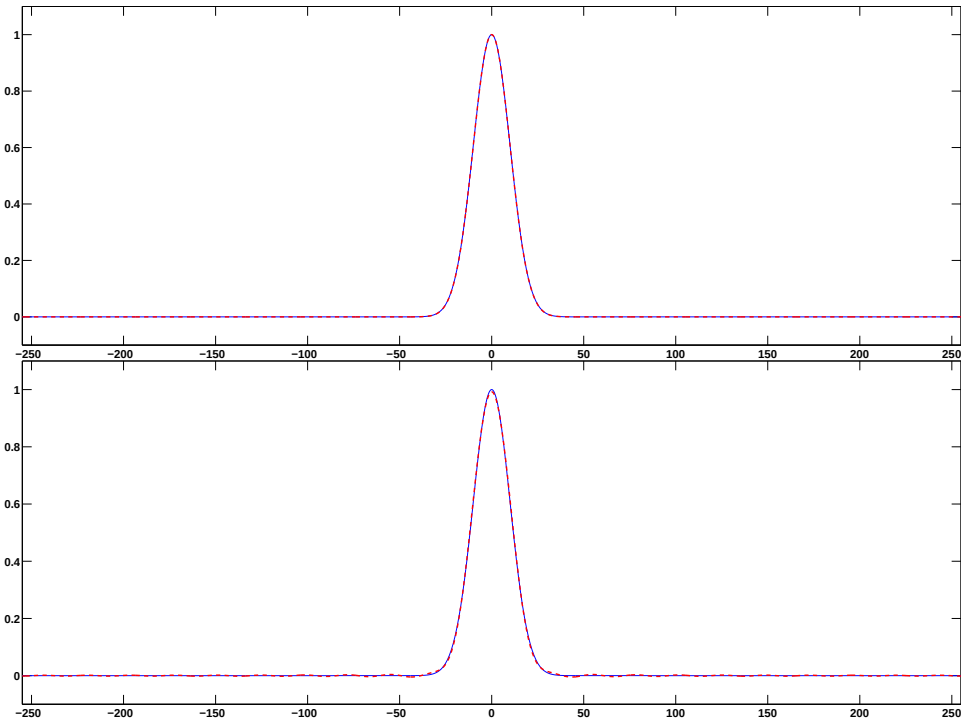


Figure 1: Approximation of the target Gaussian $g_{\sigma_r}(s)$ on the interval $[-255, 255]$ using the raised cosine $\phi(s)$ in (8). We have used $\sigma_r = 10$ in this case, for which $N_0 = 263$. **Top**: The solid blue line shows the target Gaussian $g_{\sigma_r}(s)$, and the broken red line is the raised cosine $\phi(s)$. We truncate $\phi(s)$ to obtain $\phi_\varepsilon(s)$ in (16), where we used $\varepsilon = 0.005$. In this case, $M = 111$. Thus, a total of $2 \times M = 222$ terms are dropped from the series. The truncation $\phi_\varepsilon(s)$ has only 42 terms. **Bottom**: The solid blue line is $g_{\sigma_r}(s)$, and the broken red line is $\phi_\varepsilon(s)$. Note that the quality of approximation is reasonably good even after discarding almost 84% terms. As a result of the truncation, small oscillations of size ε emerge on the tails. The approximation around the origin is positive and monotonic.

Algorithm 1 Fast algorithm for computing T

Input: Image $f(\boldsymbol{x})$ and R.
Return: T as in (10).
1. Set R as window of `MAX-FILTER`.
2. Apply `MAX-FILTER` along each row of $f(\boldsymbol{x})$; return image $m(\boldsymbol{x})$.
3. Apply `MAX-FILTER` along each column of $m(\boldsymbol{x})$; return image $M(\boldsymbol{x})$.
4. Set T as the maximum of $f(\boldsymbol{x}) - M(\boldsymbol{x})$ over all $\boldsymbol{x}$.

3 Acceleration using truncations

We have seen that, by using the exact value of T, we can bring down the run time by $10 - 20\%$. Unfortunately, Table 1 tells us that this alone is not sufficient in the regime $\sigma_r < 15$. For example, N_0 is of the order 10^3 in the regime $\sigma_r < 5$. So why do we require so many terms in (8) and (9) to approximate a narrow Gaussian? This is exactly because we are forcing $\phi(s)$ to positive and monotonic on its broad tails, where $g_{\sigma_r}(s)$ is close to zero. For example, consider the approximation in (8):

$$\phi(s) = \sum_{n=0}^{N} 2^{-N} \binom{N}{n} \cos\left(\frac{(2n-N)s}{\sqrt{N}\sigma_r}\right). \tag{14}$$

By requiring $N > N_0$, we can guarantee that (1) $\phi(s)$ is close to $g_{\sigma_r}(s)$, and (2) $\phi(s)$ is positive and monotonic over $[-T, T]$. Note, however, that $g_{\sigma_r}(s)$ falls off very fast, and almost vanishes outside $\pm 3\sigma_r$. It turns out that only a few significant terms in (14) contribute to the approximation in the $\pm 3\sigma_r$ region. The rest of the terms have a negligible contribution, and are required only to force positivity at the tails.

Table 3: The threshold N_0 before and after truncation, tolerance $\varepsilon = 0.05$ (worst-case setting $T = 255$).

σ_r	3	5	8	10	12	15
N_0 (before)	2929	1053	413	263	184	119
N_0 (after)	95	77	53	43	36	29
% of terms dropped	96	92	88	85	82	77

Thanks to expression (14), it is now straightforward to determine which are the significant terms. Note that the coefficients $2^{-N}\binom{N}{n}$ in (14) are positive and sum up to one. In fact, they are unimodal and closely follow the shape of the target Gaussian. The smallest coefficients are at the tails, and

the largest coefficients are at the center. In particular, the smallest coefficient is $1/2^N$, while the largest one is $2^{-N}[(N/2)!]^{-2}N!$ (assuming N to be even). For large N, the latter is approximately $\sqrt{2/\pi N}$ using Stirling's formula. Thus, as N gets large, the coefficients get smaller. What is perhaps significant is that the ratio of the smallest to the largest coefficient is $(\pi N 2^{2N-1})^{-1/2}$, and this keeps shrinking at an exponential rate with N. On the other hand, the cosine functions (which act as the interpolating function) are always bounded between $[-1, 1]$.

The above observation suggests dropping the small terms on the tail. In particular, for a given tolerance $\varepsilon > 0$, let $M = M(\varepsilon)$ be the smallest term for which

$$\sum_{n=0}^{M+1} 2^{-N} \binom{N}{n} > \varepsilon/2, \tag{15}$$

and set

$$\phi_\varepsilon(s) = \sum_{n=M}^{N-M} 2^{-N} \binom{N}{n} \cos\left(\frac{(2n-N)s}{\sqrt{N}\sigma_r}\right). \tag{16}$$

It follows that the error $|\phi(s) - \phi_\varepsilon(s)|$ is within ε for all $-T \le s \le T$. Note that $\phi_\varepsilon(s)$ is symmetric, but is no longer guaranteed to be positive on the tails, where oscillations begin to set in. However, what we can guarantee is that the negative overshoots are within $-\varepsilon$. In fact, the quality of the final approximation turns out to be quite satisfactory. This is illustrated with an example in Figure 1. The main point is that, in the regime $\sigma_r < 15$, we are now able to bring down the order to well within 100. We list some of them in Table 3. Notice that we can drop more terms for a given accuracy as the kernel gets narrow. For $\sigma_r < 5$, we can drop almost 95% of the terms, while keeping the error within 0.5% of the peak value.

Note that (15) actually requires us to compute a large number of tails coefficients, which are eventually not used in (16). It is thus better to estimate M when N is large. A good estimate of (15) is provided by the Chernoff bound for the binomial distribution [22], namely,

$$\sum_{n=0}^{M} 2^{-N} \binom{N}{n} \le \exp\left(-\frac{(N-2M)^2}{4N}\right).$$

It can be verified the estimate is quite tight for $N > 100$. By setting the bound to $\varepsilon/2$, we get

$$M = \frac{1}{2}\left(N - \sqrt{4N\log(2/\varepsilon)}\right). \tag{17}$$

Algorithm 2 Improved SHIFTABLE-BF

Input: Image $f(\boldsymbol{x})$, variances σ_s^2 and σ_r^2, and tolerance ε.
Return: Bilateral filtered image $\tilde{f}(\boldsymbol{x})$.
1. Set R to some factor of σ_s (determined by size of spatial Gaussian).
2. Input $f(\boldsymbol{x})$ and R to Algorithm 1, and get T.
3. Set $N = 0.405(T/\sigma_r)^2$.
4. Assign M using the following rule:
   (a) [$\sigma_r > 40$] Set $M = 0$.
   (b) [$10 < \sigma_r \leq 40$] Compute M from (15), given N and ε.
   (c) [$\sigma_r \leq 10$] Plug N and ε into (17) to get M.
5. Use N and M to specify $\phi_\varepsilon(\boldsymbol{x})$ in (16).
6. Using $\phi_\varepsilon(\boldsymbol{x})$ as the range kernel, input $f(\boldsymbol{x})$ to SHIFTABLE-BF to get $\tilde{f}(\boldsymbol{x})$.

The final algorithm obtained by combining the proposed modifications is given in Algorithm 2. Henceforth, we will continue to refer to this as the SHIFTABLE-BF. The Matlab implementation of SHIFTABLE-BF can be found here [20].

4 Experiments

We now provide some results on synthetic and natural images to understand the improvements obtained used our proposal. While all the experiments were done on Matlab, we took the opportunity to report the run time of a multithreaded Java implementation of Algorithm 2. All experiments were run on an Intel quad core 2.83 GHz processor.

4.1 Run time

First, we tested the speedup obtained using Algorithm 1 for computing T. We used a Matlab implementation of this algorithm [20]. It is expected that the run time remain roughly the same for different σ_s. As seen in table 4, this is indeed the case. The run time of the direct method, for the same image and the same settings of σ_s, was already provided in table 2. Note that we have been able to cut down the time by a few orders using our fast algorithm.

We then compared the run times of multithreaded Java implementations of SHIFTABLE-BF proposed in [19] and its present refinement. For this, we

Table 4: Average time required to compute the exact value of T using Algorithm 1. We used the standard 512×512 *Lena*. See also Table 2.

σ_s	1	3	5	10	15	20	30
Time (millisec)	70	71	73	71	70	72	71

Table 5: Comparison of the run times of the multithreaded Java implementations of `SHIFTABLE-BF` (as in [19]) and its proposed improvement, for different values of σ_r (fixed $\sigma_s = 15$). We used the test image *Checker* shown in Figure 2. Shown in the table are the different ε used in Algorithm 2. We use ∞ to signify that the run time is impracticably large.

σ_r	5	8	10	12	15	20
`SHIFTABLE-BF` (millisec)	∞	∞	∞	2130	1280	800
Improved `SHIFTABLE-BF` (millisec)	880	500	350	270	250	150
ε (% of peak value)	3	2	2	1	1	1

used the test image *Checker* shown in Fig. 2. We have used small values of σ_r, and a fixed $\sigma_s = 15$. The average run times are shown in Table 5. The tolerance ε used for the truncation are also given. We use a smaller ε (larger truncation) as σ_r gets small. Notice how we have been able to cut down the run time by more than 70%. This is not surprising, since we have discarded more than 85% of terms. Notice that the run times are now well within 1 second. The run time of the direct implementation (which does not depend on σ_r) was around 10 seconds. The main point is that we can now implement the filter in a reasonable amount of time for small σ_r, which could not be done previously in [19].

4.2 Accuracy

We next studied the effect of truncation. To get an idea of the noise that is injected into the filter due to the truncation, we used the *Checker* image. This particular image allowed us to test both the diffusive and the edge-preserving properties of the filter at the same time. We used the setting $\sigma_s = 30$ and $\sigma_r = 10$. First, we tried the direct implementation of (1), using a very fine discretization. Then we tried Algorithm 2. The difference between the two outputs is shown in Figure 3. The artifacts shown in the image are actually quite insignificant, within 10^{-5} times the peak value. Notice that most of the artifacts are around the edges. This comes from the oscillations induced at the tails of kernel by the truncation. To compare the filter outputs

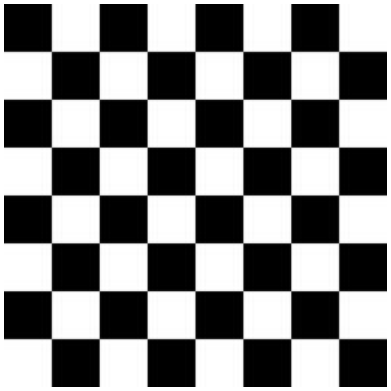


Figure 2: Test image *Checker* of size 256×256 consisting of black (intensity 0) and white (intensity 255) squares. The bilateral filter acts as a diffusion filter in the interior of the squares, and as an edge-preserving filter close to the boundary. It preserves both the constant-intensity regions, and the jumps across squares.

(with and without truncation), we extracted two horizontal scan profiles from the respective outputs. These are shown in Figure 4. Notice that it is rather hard to distinguish the two.

We then applied the filters on the standard grayscale image of *Lena*. We first applied the direct implementation followed by Algorithm 2. In this case, T was computed to be 215. The difference image is shown in Figure 5. It is again seen that the small artifacts are cluttered near the edges. We have also tried measuring the mean-squared-error (MSE) for different σ_r. The results are given in Table 6. Note that relatively larger MSEs are obtained at small σ_r. This is because we are forced to use a large truncation to speed up the filter at small σ_r. The above results show that we can drastically cut down the run time of filter using the proposed modifications, without incurring significant errors.

Table 6: The mean-squared-error (MSE) between the filter outputs before and after truncation. We use the *Lena* image, and a fixed $\sigma_s = 30$. The tolerance ε is chosen as in Table 5.

σ_r	5	8	10	12	15	20
$10\log_{10}(\mathrm{MSE})$	-9.3	-11.1	-11.8	-13.5	-13.8	-14.1

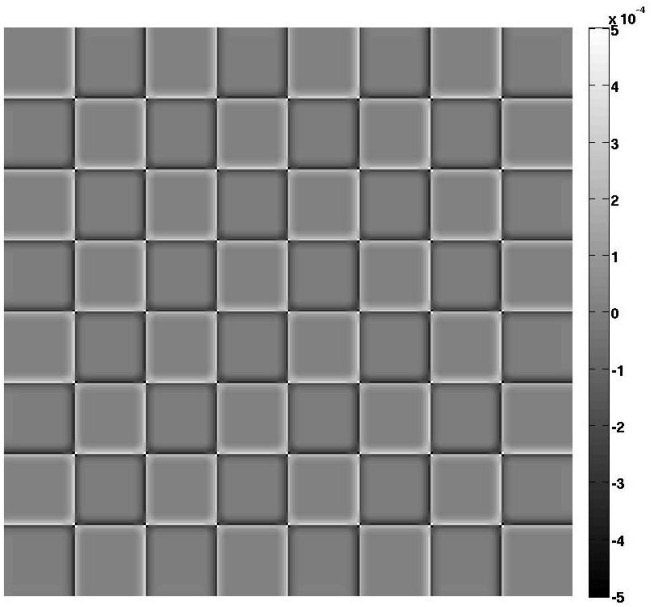
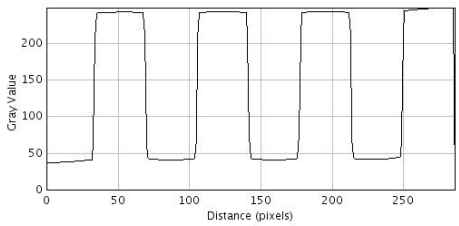
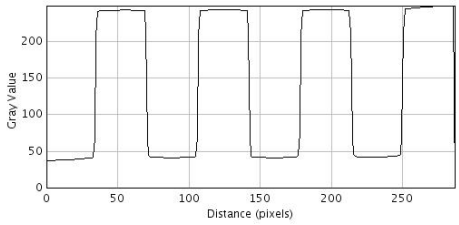


Figure 3: The difference between the outputs of the direct (high resolution) implementation of (1), and that obtained using Algorithm 2. The test image in Figure 2 was used as the input, and settings were $\sigma_s = 30$ and $\sigma_r = 10$. The noise created due to the truncation is actually very small – the error is within 10^{-5} times the peak value. See the comparison of scan profiles in Figure 4.



(a) High resolution implementation.



(b) Our implementation.

Figure 4: Comparison of the respective scan profiles from the bilateral filter outputs (cf. description in Figure 3).

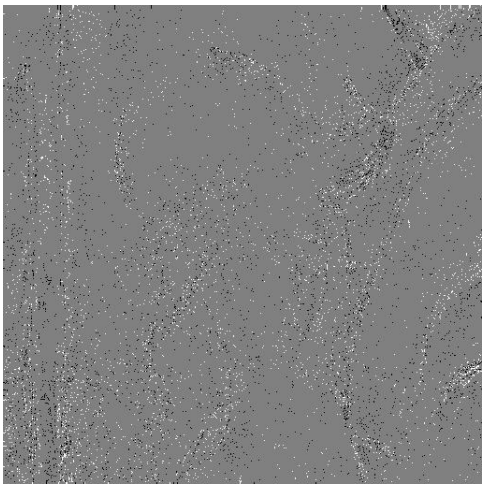


Figure 5: The difference between the outputs for the image *Lena* (size 512×512). The settings were $\sigma_s = 30$ and $\sigma_r = 10$. The noise created due to the truncation is within 10^{-4} times the peak value, and is thus practically insignificant. Notice that the errors are mainly around the edges.

4.3 Comparison with a benchmark algorithm

We next compared the performance of the improved SHIFTABLE-BF algorithm with those proposed in [14]. The latter algorithms are considered as benchmark in the literature on fast bilateral filtering. Porikli proposed a couple of algorithms in [14] – one using a variable spatial filter and a polynomial range filter (we call this BF1), and the other using a constant spatial filter and a variable range filter (we call this BF2). The difficulty with BF1 is that it is rather difficult to control the width of the polynomial range filter. In particular, as was already mentioned in the introduction, it is difficult to approximate narrow Gaussian range kernels using BF1. We refer the interested readers to the experimental results in [19], where a comparison was already made between BF1 and SHIFTABLE-BF. For completeness, we perform a single experiment to compare these filters when σ_r is small (a rather large value of σ_r was used in the experiments in [19]). For this, and the remaining experiments, we will consider the standard test image of *Barbara* of size 512×512. This image has several texture patterns, and is well-suited for comparing the performance of bilateral filters with narrow range kernels[1]. In particular, we consider the Gaussian bilateral filter with $\sigma_s = 20$ and $\sigma_r = 20$. The results obtained using SHIFTABLE-BF and BF1 are shown in Figure 6. The error between the direct implementation of the bilateral filter and SHIFTABLE-BF was within 10^{-3}. On the other hand, note how BF1 completely breaksdown. The reason for this was already mentioned in the introduction. A similar breakdown, with a larger σ_r, was also observed in Figure 3 in [19].

We next considered BF2, which does not suffer from the above problem. However, we note that BF2 cannot be used to perform Gaussian bilater filtering – it only works with constant spatial filters (box filters). This is because BF2 uses fast integral histograms, and this only works for box filters. To make the comparison even, we considered bilateral filters with constant spatial filters and Gaussian range kernels. We note SHIFTABLE-BF can be trivially modified to work with arbitrary spatial filters.

First, we compared the run times of the Matlab implementations of SHIFTABLE-BF and BF2. The results obtained at particular settings of σ_s (radius of box filter) and σ_r are shown in Figure 7. We see that the run time of SHIFTABLE-BF is consistently better than that of BF2. The difference is particularly large when $\sigma_r > 10$, and it closes down as σ_r gets small. All these can be perfectly explained. Note that, as per the design, the computational complexity of BF2 is $O(1)$ both with respect to σ_s and σ_r.

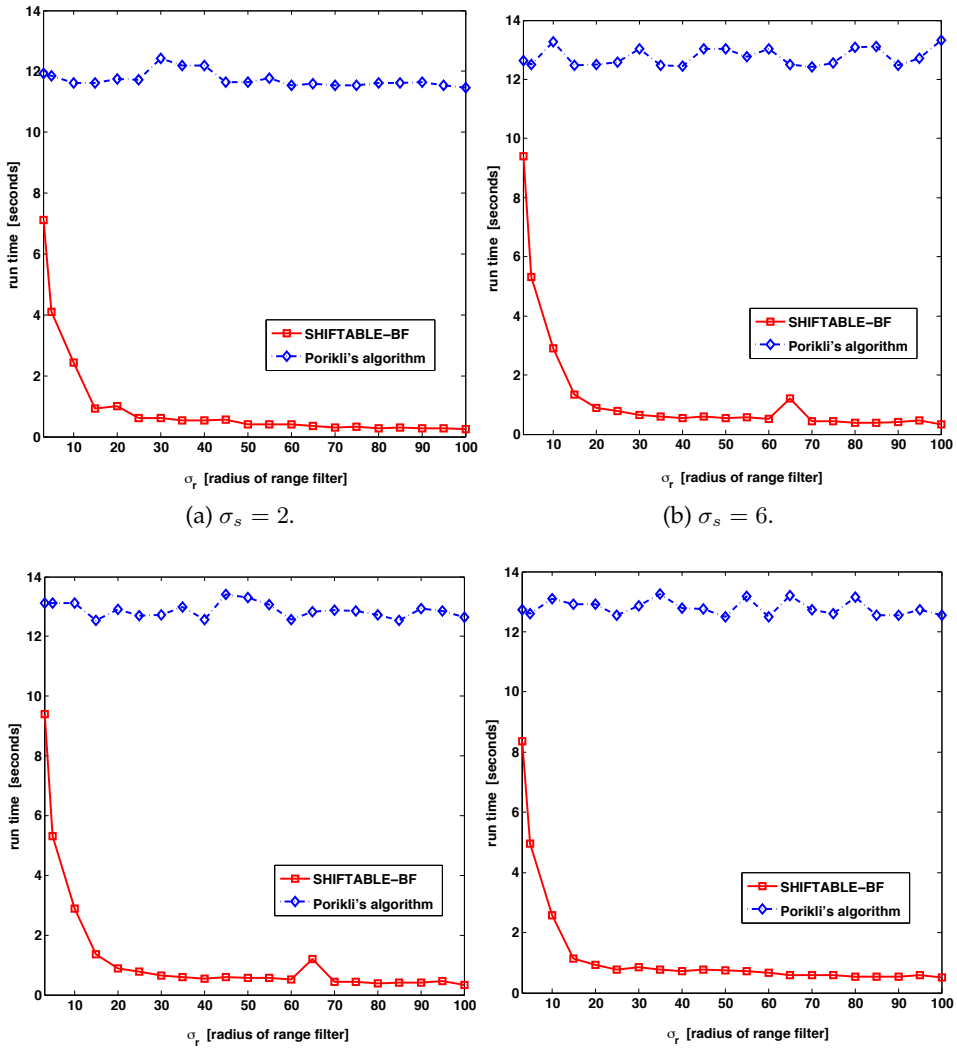
[1] we thank one of the reviewers for suggesting this example.

(a) Output of SHIFTABLE−BF. (b) Output of Porikli's algorithm.

Figure 6: Comparison of the bilateral filtering results obtained using SHIFTABLE−BF and Porikli's polynomial-kernel algorithm [14]. In both cases, we used a Gaussian spatial filter ($\sigma_s = 20$) and a Gaussian range filter ($\sigma_r = 20$). For our algorithm, we set $\varepsilon = 0.03$. For Porikli's algorithm, we used a Taylor polynomial with order comparable to that of the raised cosine kernel.

This is indeed seen to be the case from the run times. On the other hand, the computational complexity of SHIFTABLE−BF is $O(1)$ with respect to σ_s (this is again clear from the plots in Figure 7). However, for a given σ_s, the complexity of the the original algorithm scales as $O(1/\sigma_r^2)$. The complexity, in fact, remains roughly the same even after the proposed truncation. This explains the step rise in the run time for small values of σ_r, as shown in Figure 7. However, the actual run time goes down substantially as a result of the truncations (cf. Table 5). In particular, we have noticed that the worst case run time of SHIFTABLE−BF is less than the average run time of BF2 for σ_r as low as 3.

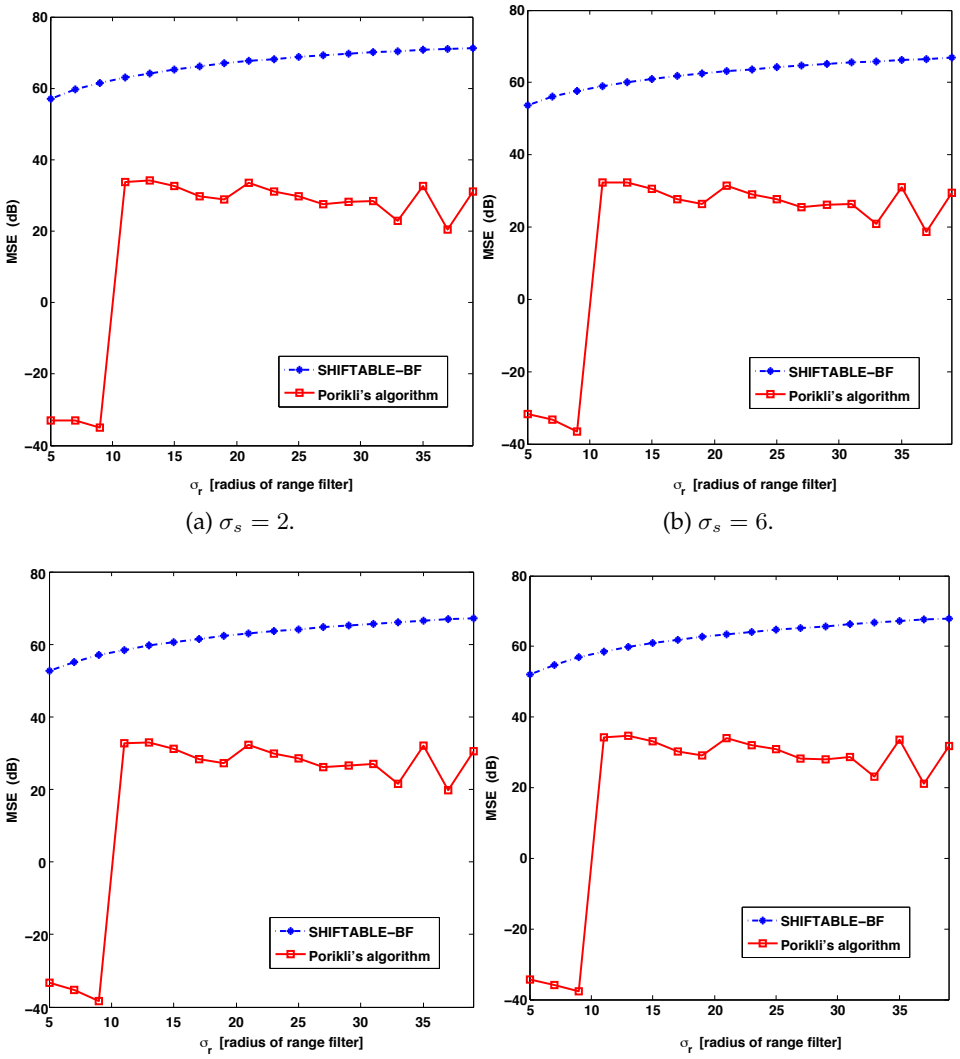
We note that the run time of BF2 depends on the number of bins used for the integral histogram. In the above experiments, we used as many bins as the grayscale levels of the image. It is thus possible to reduce the run time by cutting down the resolution of the histogram. However, this comes at the cost of the quality of the filtered image. This lead us to compare the outputs of SHIFTABLE−BF and BF2. In Figure 8, we compared the MSEs of the two algorithms for different σ_s and σ_r for the image *Barbara*. We note that the MSE for SHIFTABLE−BF is significantly lower than BF2. The gap is around 30 dB for $\sigma_r > 10$, and around 90 dB when $\sigma_r \leq 10$. We noticed that this



(a) $\sigma_s = 2$. (b) $\sigma_s = 6$.

(c) $\sigma_s = 10$. (d) $\sigma_s = 14$.

Figure 7: Comparison of the run times for different σ_s and σ_r. We compare our algorithm `SHIFTABLE-BF` with Porikli's algorithm `BF2` (using integral histograms [14]). In either case, we use a constant spatial filter and a Gaussian range filter. For our algorithm, we set $\varepsilon = 0.01$. For Porikli's algorithm, we use the full resolution (256 bin) histogram. Both the algorithms were implemented in Matlab.



(a) $\sigma_s = 2$. (b) $\sigma_s = 6$.

(c) $\sigma_s = 10$. (d) $\sigma_s = 14$.

Figure 8: Comparison of the MSE for different σ_s and σ_r. The MSEs are computed between the direct implementation and `SHIFTABLE-BF`, and between the direct implementation and `BF2`. The parameter settings are identical to those used in Figure 7.

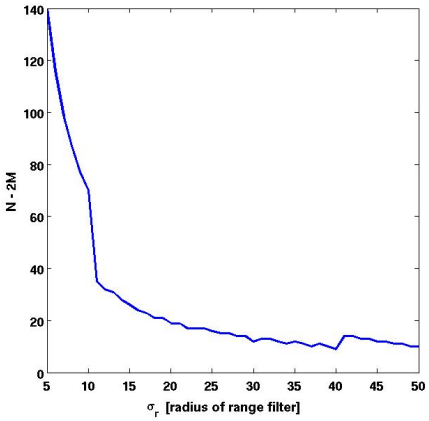


Figure 9: Dependence of order on σ_r. Here N and M are as defined in Algorithm 2. The sudden jump at $\sigma_r = 10$ is due to the truncation rule in (17).

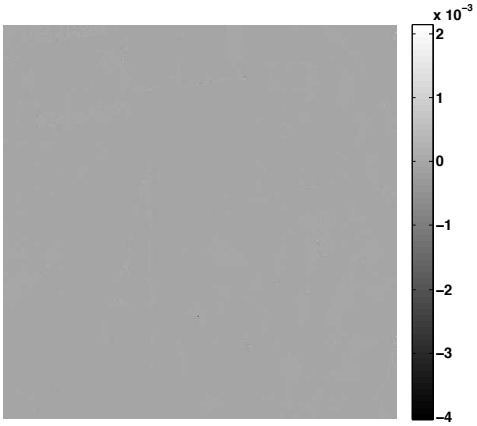
difference becomes even more pronounced if we use a smaller number of bins. As expected, note that the individual MSEs do not vary much with σ_s. The reader will notice that the MSE of `SHIFTABLE-BF` suddenly drops by 60 dB when $\sigma_r \leq 10$. To explain this, we plot the effective order $N - 2M$ for different σ_r in Figure 9. We see that the order (of approximation) suddenly jumps up when σ_r goes below 10, which explains the jump in the MSE in Figure 8. This is simply due to the rule (17) used in Algorithm 2

In Figure 10, the filtered outputs of the two algorithms are compared with the direct implementation, for a small value of σ_r. Note that the pointwise error between the direct implementation and `SHIFTABLE-BF` is of the order 10^{-3}. On the other hand, the corresponding error between the direct implementation and `BF2` is substantial, a few orders larger than that for `SHIFTABLE-BF`. This explains the large gap between the MSEs in Figure 8.

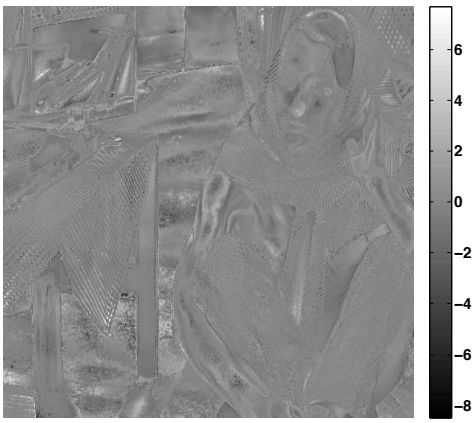
We close this section by commenting on the memory usage of `SHIFTABLE-BF` and `BF2`. The former requires us to compute and store a total of $2(2N - M)$ images, while the latter requires us to store a histogram with B bins per pixel (equivalent of B images). It is clear from Figure 9 that even for a half-resolution histogram ($B = 128$) and for $\sigma_r > 10$, the memory requirement of `BF2` is comparable to that of `SHIFTABLE-BF`. For smaller values of σ_r, `SHIFTABLE-BF` clearly requires more memory than `BF2`.

(a) Test image *Barbara* (size 512×512).

(b) Bilateral filter output (direct implementation).



(c) Error between (b) and our method.



(d) Error between (b) and Porikli's method.

Figure 10: Comparison of the results obtained using `SHIFTABLE-BF` with the direct implementation (high resolution) and Porikli's algorithm `BF2`. In all three cases, we used a constant spatial filter (radius = 20) and a Gaussian range filter ($\sigma_r = 5$). For `SHIFTABLE-BF`, we set $\varepsilon = 0.01$. For `BF2`, we used the best possible resolution (256 bin histogram). The run times of the Matlab implementations were: Direct implementation (37 seconds), `BF2` (13 seconds), and `SHIFTABLE-BF` (6 seconds).

5 Discussion

In this paper, we proposed some simple ways of accelerating the bilateral filtering algorithm proposed in [19]. This, in particular, opened up the possibility of implementing the algorithm in real-time for small σ_r. We note that the problem of determining the optimal σ_s and σ_r for a given application is extrinsic to our algorithm. We are only required to determine the parameter T, which is intrinsic to our algorithm. A fast algorithm was proposed in the paper for this purpose. However, we note that having a fast algorithm does make it easier to determine the optimal parameters. In this regard, we note that Kishan et al. have recently shown how our fast algorithm can be used to tune the parameters for image denoising, under different noise models [25, 28]. One crucial observation used in these papers is that the a certain unbiased estimator of the MSE can be efficiently computed for our fast bilater filter, using the linear expansions in (6) and (7). The "best" parameters are choosen by optimizing this MSE estimator. While this can also be done for the polynomial-based bilateral filter in [14], this trick cannot be used for other fast implementations of the bilateral filter, at least to the best of our knowledge.

Finally, we note that the ideas proposed here can also be extended to the $O(1)$ algorithm for non-local means given in [24]. In non-local means [2], the range kernel operates on patches centered around the pixel of interest. A coarse non-local means was considered in [24], where a small patch neighborhood consisting of the pixels $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p$ (where, say, $\boldsymbol{u}_1 = 0$) was used. In this case, the main observation was that formula for the non-local means can be written in terms of following sums:

$$\int_{\|\boldsymbol{y}\| \leq R} f(\boldsymbol{x}-\boldsymbol{y}) g(f(\boldsymbol{x}+\boldsymbol{u}_1) - f(\boldsymbol{x}-\boldsymbol{y}+\boldsymbol{u}_1), \ldots, f(\boldsymbol{x}+\boldsymbol{u}_p) - f(\boldsymbol{x}-\boldsymbol{y}+\boldsymbol{u}_p))\, d\boldsymbol{y}, \tag{18}$$

and

$$\int_{\|\boldsymbol{y}\| \leq R} g(f(\boldsymbol{x}+\boldsymbol{u}_1) - f(\boldsymbol{x}-\boldsymbol{y}+\boldsymbol{u}_1), \ldots, f(\boldsymbol{x}+\boldsymbol{u}_p) - f(\boldsymbol{x}-\boldsymbol{y}+\boldsymbol{u}_p))\, d\boldsymbol{y}, \tag{19}$$

where $g(s_1, \ldots, s_p)$ is an anisotropic Gaussian in p variables, and has a diagonal covariance. This looks very similar to (1), except that we now have a multivariate range kernel. By using the separability of $g(s_1, \ldots, s_p)$, and by approximating each Gaussian component by either (8) or (9), a $O(1)$

algorithm for computing (18) and (19) was developed. We refer the readers to [24] for further details. The key consideration with this algorithm is that the overall order scales as N^p, where N is the order of the Gaussian approximation for each component. In this case, it is thus important to keep N as low as possible for a given covariance. After examining (18) and (19), it is clear that the interval over which the one dimensional Gaussians need to be approximated is $[-T, T]$, where T is as defined in (10). We can compute this using Algorithm 1. Moreover, we can further reduce the order by truncation, especially when the covariance is small. However, N^p can still be large (even for $p = 3$ or 4), and hence a parallel implementation must be used for real-time implementation.

6 Appendix : MAX-FILTER algorithm

We explain how the `MAX-FILTER` algorithm works in one dimension. Let $f_1, f_2, \ldots, f_N$ be given, and we have to compute $\max(f_{i-R}, \ldots, f_{i+R})$ at every interior point i. Assume R is an integer, and N is a multiple of the window size $W = 2R + 1$, say, $N = pW$ (padding is used if this not the case). The idea is to compute the local maximums using running maximums, similar to running sums used for local averaging [21]. The difference here is that, unlike averaging, the max operation is not linear. This can be fixed using "local" running maximums.

We begin by dividing $f_1, f_2, \ldots, f_N$ into p equal partitions. The kth partition ($k = 0, 1, \ldots, p-1$) is composed of $f_{1+kW}, \ldots, f_{W+kW}$. For a given partition k, we recursively compute the two running maximums (of length W), one from from the left and one from the right. Let $l^{(k)}$ and $r^{(k)}$ be the left and right running maximums for the kth partition. The sequence $l^{(k)}$ start at the left of the partition with $l_1^{(k)} = f_{1+kW}$, and is recursively given by $l_i^{(k)} = \max(\, l_{i-1}^{(k)}, f_{i+kW}\,)$ for $i = 2, 3, \ldots, W$. It ends on the right end of the partition. On the other hand, $r^{(k)}$ start at the right and ends on the left: $r_W^{(k)} = f_{W+kW}$, and $r_{W-i}^{(k)} = \max(\, r_{W-i+1}^{(k)}, f_{W-i+kW}\,)$ for $i = 1, 2, \ldots, W-1$. This is done for every partition to get $l^{(0)}, \ldots, l^{(p-1)}$ and $r^{(0)}, \ldots, r^{(p-1)}$.

We now concatenate the left maximums into a single function $l_1, \ldots, l_N$, that is, we set $l = (l^{(0)}, \ldots, l^{(p-1)})$. Similarly, we concatenate the right maximums in order, $r = (r^{(0)}, \ldots, r^{(p-1)})$. In practice, we just need to recursively compute $l_1, \ldots, l_N$ and $r_1, \ldots, r_N$, resetting the recursion at the boundary of every partition. We now split $f_{i-R}, \ldots, f_{i+R}$ into two segments, which either belong to the same partition or two adjacent parti-

tions. Then, from the associativity of the max operation, it is seen that $\max(f_{i-R}, \ldots, f_{i+R}) = \max(r_{i-R}, l_{i+R})$. Note that, we need just 3 max operations per point to get the result, independent of the window size R.

7 Acknowledgments

This work was partly supported by the Swiss National Science Foundation under grant PBELP2-135867. The author thanks M. Unser and D. Sage for interesting discussions, and the anonymous referees for their helpful comments and suggestions. The author also thanks A. Singer and the Program in Applied and Computational Mathematics at Princeton University for hosting him during this work.

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," Proc. IEEE International Conference on Computer Vision, pp. 839-846, 1998.

[2] A. Buades, B. Coll, and J.M. Morel, "A review of image denoising algorithms, with a new one," Multiscale Modeling and Simulation, vol. 4, pp. 490-530, 2005.

[3] L.P. Yaroslavsky, *Digital Picture Processing–An Introduction*, Springer-Verlag, Berlin, 1985.

[4] E.P. Bennett, J.L. Mason, and L. McMillan, "Multispectral bilateral video fusion," IEEE Transactions on Image Processing, vol. 16, pp. 1185-1194, 2007.

[5] H. Winnemoller, S. C. Olsen, and B. Gooch, "Real-time video abstraction," ACM Siggraph, pp. 1221-1226, 2006.

[6] R. Ramanath and W. E. Snyder, "Adaptive demosaicking," Journal of Electronic Imaging, vol. 12, pp. 633-642, 2003.

[7] M. Mahmoudi and G. Sapiro, "Fast Image and Video Denoising via Nonlocal Means of Similar Neighborhoods," IEEE Signal Processing Letters, vol. 12, no. 12, pp. 839-842, Dec. 2005.

[8] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi, "Bilateral filtering-based optical flow estimation with occlusion detection," European Conference on Computer Vision, pp. 211-224, 2006.

[9] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister, "Stereo matching with color-weighted correlation, hierarchical belief propagation and occlusion handling," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, pp. 492-504, 2009.

[10] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "Bilateral Filtering: Theory and Applications," Foundations and Trends in Computer Graphics and Vision, vol. 4, no. 1, pp. 1-73, 2009.

[11] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," ACM Siggraph, vol. 21, pp. 257-266, 2002.

[12] B. Weiss, "Fast median and bilateral filtering," ACM Siggraph, vol. 25, pp. 519-526, 2006.

[13] Q. Yang, K.-H. Tan, and N. Ahuja, "Real-time $O(1)$ bilateral filtering," IEEE Conference on Computer Vision and Pattern Recognition, pp. 557-564, 2009.

[14] F. Porikli, "Constant time $O(1)$ bilateral filtering," IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-8, 2008.

[15] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," European Conference on Computer Vision, pp. 568-580, 2006.

[16] K. N. Chaudhury, A. M.-Barrutia, and M. Unser, "Fast space-variant elliptical filtering using box splines," IEEE Transactions on Image Processing, vol. 19, pp. 2290-2306, 2010.

[17] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," Pattern Recognition Letters, vol. 13, no. 7, pp. 517-521, 1992.

[18] J. Gil and M. Werman, "Computing 2-d min, median, and max filters," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 5, pp. 504-507, 1993.

[19] K. N. Chaudhury, Daniel Sage, and M. Unser, "Fast O(1) bilateral filtering using trigonometric range kernels," IEEE Transactions on Image Processing, vol. 20, no. 12, pp. 3376-3382, 2011.

[20] K. N. Chaudhury, *Fast Bilateral Filter* (www.mathworks.com/matlabcentral/fileexchange/36657), MATLAB Central File Exchange, retrieved May 21, 2012.

[21] P. S. Heckbert, "Filtering by repeated integration," International Conference on Computer Graphics and Interactive Techniques, vol. 20, no. 4, pp. 315-321, 1986.

[22] N. Alon and J. Spencer, *The Probabilistic Method*, Wiley-Interscience, 2000.

[23] F. C. Crow, "Summed-area tables for texture mapping," ACM Siggraph, vol. 18, pp. 207-212, 1984.

[24] K. N. Chaudhury, "Constant-time filtering using shiftable kernels," IEEE Signal Processing Letters, vol. 18, no. 11, pp. 651-654, 2011.

[25] H. Kishan and C. S. Seelamantula, "SURE-Fast bilateral filters," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2012.

[26] B. K. Gunturk, "Fast Bilateral Filter With Arbitrary Range and Domain Kernels," IEEE Transactions on Image Processing, vol. 20, no. 9, pp. 2690-2696, 2011.

[27] A. Adams, J. Baek, M. A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," Computer Graphics Forum, vol. 29, no. 2, pp. 753-762, 2010.

[28] H. Kishan and C. S. Seelamantula, "Optimal parameter selection for bilateral filter using Poisson unbiased risk estimate," accepted in IEEE International Conference on Image Processing (ICIP), 2012.